Montage: An X-Based Multimedia Electronic Mail System

W. Keith Edwards

Graphics, Visualization, & Usability Center - Multimedia Computing Group
Georgia Institute of Technology
Atlanta, GA 30332-0280
keith@cc.gatech.edu

ABSTRACT
This paper describes an extensible multimedia electronic mail system called Montage which is based on the X Window System. Montage supports the composition, transmission, and viewing of structured documents consisting of virtually any type of medium. Further, users can at runtime extend the system easily to support new document types, including text, images, audio, video, executable programs, and commercial file formats.

KEYWORDS: Multimedia, electronic mail, X Window System.

INTRODUCTION: WHY MULTIMEDIA MAIL?

In the past decade, the proliferation of fast, inexpensive, networked computer workstations has produced an explosion in the use of electronic mail. Electronic mail systems have traditionally been limited to the transmission of simple textual information.

More recently however, as computer workstations have increased dramatically in power and as the use of windowing interfaces becomes more widespread and standardized, the capabilities have emerged for the composition, transmission, and reception of complex multimedia electronic mail messages consisting of voice, imaged, video, and other media (in addition to plain text of course).

The chief advantage of a multimedia electronic mail system is the increased bandwidth between the users of the system. Research indicates that some concepts may be most appropriately expressed in certain media. [7, 8, 10] A good multimedia mail system should allow people to communicate with one another as freely and without restriction as conventional “paper” mail systems do. With paper mail, users can compose documents, jot notes onto them, and then seal them along with any other enclosures into an envelope and expect prompt delivery. Multimedia mail systems would offer similar flexibility.

Graphics, Visualization, and Usability Center Technical Report GIT-GVU-92-29, October 1992. Georgia Tech.

Furthermore, since the most basic goal of any electronic mail system is to expedite and enhance the communication between people, care should be taken to conform to mail transport standards. An electronic mail system which does not interoperate with a broad range of systems denies its users communication with those systems.

This paper presents a multimedia electronic mail system called *Montage* which has been developed at the Georgia Tech Multimedia Computing Group (a part of the Graphics, Visualization, and Usability Center under the direction of Dr. James Foley) [6]. We believe that this system provides a flexible and convenient means to send and receive complex multimedia documents. This system is built strictly on top of lower-level mail transport standard and thus should be portable to and interoperate with many systems. Our current implementation is on the Unix operating system and X11, specifically Motif 1.1.

DESIGN GOALS AND HISTORY

When work was started on this project late in 1990 [5], we began with several principles that we hoped would lead to a powerful and flexible system. This section describes our three major design goals.

Extensibility

First, and perhaps most importantly, we wanted to build a system that would not be a “closed box.” That is, we wanted a system which could be easily extended by its users at runtime to support arbitrary media. Too many email systems support only a restricted range of media. However complex these media may be, there is still no provision for extending the system.

At the time we began our work, there were several multimedia mail systems already available for Unix systems, the best-known of these being BBN Slate, the Andrew Message System, and NeXTmail. Each of these suffers from its own set of problems. For example, the BBN Slate system provides users with a number of tools for composing complex mail messages. BBN Slate messages can even

contain spreadsheet data, but users are restricted to working with the built-in, integrated Slate spreadsheet rather than the tools there are most used to.

Another system, the CMU Andrew Message System, was built using the Andrew tools provided by CMU [1]. This mail system can send any type of construction which can be expressed by the objects in the Andrew environment. Thus, the system is extensible but it can not interoperate well with systems not built with Andrew.

A third system, NeXTmail, bundled with all NeXT computers, could compose and send fairly complex documents consisting of formatted (rich) text, images, sounds, and typed files. There were (and are) a number of problems with the NeXT mail system though. First, the system is built to a large extent on primitives provided by the NextStep environment which are not likely to be found on other systems. Second, the specification for the mail encoding format (specifically the header lines used and required by the system) is not publicly available.

Mail Transport Protocols

In addition to our goal of extensibility, we wanted to construct a system which was built on top of existing standards. For our platform, this meant the Simple Mail Transport Protocol (SMTP) as a base [4, 11]. SMTP is widespread among the Unix community. As long as our mailer spoke SMTP its messages would be transmittable by the large number of mail transport agents in the world that speak SMTP.

One of the limitations of SMTP is that it only supports the transfer of non-binary data. Thus, Montage must perform a "packing" and "unpacking" process to convert the message body data to a form which can be transmitted via SMTP.

We will discuss our approaches to the problems of mail transport protocols and packing formats shortly.

Message Presentation Format

A third requirement of our system was that it should present its messages in an intuitive and easy-to-understand format. It was our belief that, unlike more general hypermedia documents, mail messages are typically generated by their authors to convey some small number of important ideas or data. Thus, whereas hypermedia documents which are reader-driven[9], mail messages are typically author-driven. We reasoned that a hypertext-like presentation format may not be the most useful or intuitive for a mail system.

While we did not want our messages to be full-blown hypermedia documents, we did acknowledge that there was a need for some interactiveness within a message. One common example of this may be a document which is being distributed for review by a number of commentors or coauthors. One would like to be able to view the original document as well as selectively viewing annotations by the various reviewers. We felt that some degree of interactivity would empower the mail system and its users.

As we shall discuss later, we feel that the restriction against generalized hypertext has had a simplifying effect on the design and implementation of the mail system. We also feel that the interactiveness provides a great deal of the system's power.

The next three sections address our solutions to these goals as they are currently embodied in Montage.

A MODEL FOR MULTIMEDIA MESSAGES

We mentioned that one of our design goals was to develop a presentation format for multimedia messages that was (1) somewhat more restricted than hypermedia systems to facilitate the type of communication common in electronic mail, and (2) allowed some degree of interactivity, especially in support of annotations.

Our model for multimedia mail messages essentially consists of two parts. First, all messages consist of a main body (called the *primary part*). The primary part consists of any number of components (called *chapters*) which may themselves be of any media type. All of the chapters of the primary part appear in linear order, just like a single paper document. Montage presents the primary part in a scollable window.

In addition to the primary part, a Montage message may also have zero or more *attachments*. Attachments are analogous to margin notes or "Post-It" notes in a paper document. They allow the author to attach supplemental information which refers to or supports the original document. In Montage, attachments appear as small icons on the border of the primary part. The image in the icon denotes the type of medium in the attachment; the attachment is activated or opened by clicking on it with the mouse. At composition time the author chooses the spatial location of the attachment so that, for example, comments to a document can be located across from the portion of the document they refer to. As the main body of the message is scrolled, the attachments scroll so that they keep their relative position to the part of the main body they refer to. Just like chapters in the primary part, attachments can be of any media type. Figure 1 shows a sample Montage mail message in the system mail viewer. The text and graphics compose the primary part of the message. The small phonograph icon denotes an audio attachment which is spatially located across from the charts.

There are a great number of uses for this scheme of attachments to messages. One of the canonical examples which we have already referred to is coauthoring and commenting of written documents. In Montage, a document could be exchanged in mail as the primary part of a message.

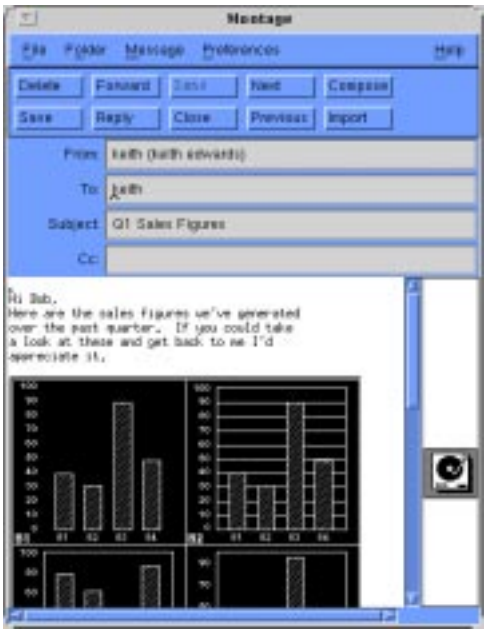

FIGURE 1. An example Montage message

Various authors can then attach comments, rewrites, graphics, audio annotations, or even video clips to the document at appropriate points. Research supports the fact that some types of revisions are most appropriate in non-textual media [9].

Also, since attachments provide a means of having an *active* message which can interact with the user, several uses for this model which are not based on simply annotation of documents come to mind. For example, a mail message may contain a large number of spreadsheets, say, one for every operating month of a company's history. The recipient of this mail message may not want to view data from every month, so the preferred format may be a message with an attachment for every spreadsheet which may then be opened at the reader's discretion.

Another example may be an attachment which consists of a shell script to upgrade some software package on the user's machine. The main body may have the message "Click here to upgrade to release 2.0." Across from that is the attachment which will perform the upgrade when clicked. (We are ignoring the security problems involved in sending actual executable programs through the mail for now. Obviously you would not want to execute a program sent to you through the mail unless you trusted the source of the message and could verify that indeed the message had come from that source).

The important thing is that the author has the choice at composition time to decide on the layout of the message. We feel that this format gives us a substantial amount of power in a relatively easy to express (and implement) fashion.

EXTENSIBILITY AND CONFIGURABILITY

Ideally, a multimedia mail system should be able to transport virtually *any* medium. This includes not only text and graphics, but also various dynamic media (such as audio, video, animation, and even executable programs), and various commercial file formats (used by spreadsheets, desktop publishing packages, and so on).

Obviously it is not feasible for the builders of the mail system to have to compile support for these media into the mailer itself. This requires a great deal of work on the part of the system builders to maintain support for all of these various formats, and also means that if the system builders don't choose to support a given format, the users of that format will be out of luck. Further, the mail system is always a "step behind" the rest of the applications world: the mail system builders are continually playing "catch up" to build in support for new formats as they become available.

Perhaps the best solution to the extensibility problem is to have a computing environment in which applications can communicate with one another, and application objects can be shared among and embedded in different applications. Some strongly object-oriented environments (NeXTstep comes to mind) do support this type of behavior, but it is not available yet in the X world.

Since this solution wasn't available to us, we had to take another approach to solving the extensibility problem. Our solution is to externalize as much of the work of interpreting and handling the various media outside the mail system as possible. We use external programs (called *handlers*) that "understand" the various media to display and edit them. In the Montage model, the mailer itself is very simple; it is basically just a framework which has the responsibility of parsing the mail messages, providing the basic mailer functionality (folders, aliases, and so on), invoking the external handlers, and creating a nice presentation for the overall message. The work of understanding what a particular medium "means" and then doing the right thing with it is solely the responsibility of the external handlers.

Media types are identified by *tags* which are simple ASCII strings that are sent along with the message components when it is transmitted (see the section Transport Protocols and Packing Formats). Montage itself associates no meaning with the media tags; instead it decides which handler to invoke on a particular component by looking up its tag in a per-user database which maps tags to handler programs chosen by the user.

This design has several benefits. First, users can make use of the applications most familiar to them to view and edit message components. Vi and Emacs users will be able to choose their favorite editor to compose text components. Secondly, if a work group begins to use a new application for its work, it is easy to enable support for the new application's data format by simply assigning it a tag and putting an entry in the database that specifies the program the be run when a message component with this tag is encountered.

Because of the presentation format we are using (described in the previous section), message components belonging to the primary part of the message are presented "in-line" (that is, they visually and structurally form a single document, rather than being presented in different windows), while attachment components are presented outside the main body of the message in their own windows when they are activated.

The different requirements of displaying message components in-line and outside of the main flow of the document require us to have the notion of several classes of handlers. The basic handlers are:

- **Editor** The program which will be invoked when the user wishes to compose a message component of a given type.
- **Renderer** The program which will be invoked when a message component needs to be displayed in-line to a message.
- **Viewer** The program which will be invoked when a message component needs to be displayed outside of a document.

The editor and viewer handlers operate as expected. They are the normal applications found on a system which are used to display and edit application-specific data (such as Lotus 1-2-3 or FrameMaker or a paint program). These applications, by default, create their own windows as children of the X root window. Thus, when invoked by Montage they will start up and appear as top-level windows "outside" the message itself. Thus, the viewer handler is the program which will be invoked whenever an attachment is opened.

But what about message components which should be displayed in the main body of the message? By default, most any applications will create their own top-level windows when they start up. We need to be able to display message data inside the main body of the message as well as in separate windows. Our solution to this is the notion of renderers. Renderers are programs which know how to draw (or "render") the media type *inside* the main body of the message.

Since this is a rather unusual requirement, most systems will not have renderers already on them waiting to be used (as is the case with viewers and editors, which are conventional off-the-shelf applications). We are working to build several renderers for common formats, and hope that if Montage ever reaches some degree of popularity there will be no shortage of publicly-available renderers.

The mechanics of how a renderer performs its job, that is, how it draws inside the main body of the mail message, are somewhat difficult. We have investigated two possible solutions. In the first potential solution, the renderer draws the medium into a pixmap and then returns the identifier of the pixmap to Montage which determines the geometry of the pixmap and copies it into the message display area. This effectively disallows any type of dynamic medium in which the contents of the displayed are subject to change.

In the second solution, which is the one we are currently working with, Montage launches renderers with a command line argument which is the window ID into which the renderer should draw. This allows fully dynamic media, but has several drawbacks. One is that the renderer must continue running as long as the message is displayed even if it is rendering a static medium. This is so that it can handle exposures in the window. Another drawback is that most existing widgets don't perform well when their windows are resized from some external controlling process. We are investigating writing a new widget which exhibits the proper behavior.

Note that if the X Toolkit provided support for forcing an application's top-level window to be specified on the command line or via some other mechanism, then we could actually embed running applications in the mail message itself. Unfortunately no X toolkit that we are familiar with provides this capability.

While the invocation of an external program for every type of media provides a great deal of flexibility, it is not the most efficient way to work with very common media which will be used on a day-to-day basis. For this reason, Montage has a few very simple handlers built in to it. Users can specify "PrimaryTextRenderer" in the configuration database to tell Montage that a tag represents simple text and that the system should use its own internal text renderer to display it. There is also a "PrimaryImageRenderer" built-in handler that can understand and display a good number of image formats internally (including Sun raster images, PBM, PPM, PGM, Gif, Faces, XWD, Group 3 FAX, MacPaint, XPM, XBM and a few others). Similarly, there is a "PrimaryTextEditor" which tells the system to use its built-in-line text editor so that users will not have to open another window to simply compose a text message.

These built-in handlers provide a certain common ground of media types that Montage can handler "out of the box" without requiring any sort of external mechanisms. Essentially they are an escape hatch around the requirement that users must have external handlers for all the media types that they wish to mail or view.

TRANSPORT PROTOCOLS AND PACKING FORMATS

As we mentioned, at the time we began work on Montage there were widely accepted standards for multipart multimedia mail transport. We knew that we must base our system on SMTP to allow interoperation with the majority of existing Unix mail agents, but beyond that there were few accepted standards. The current implementation of Montage uses a message transport format developed in-house to support our model of multimedia mail messages. In the past year, however, a format called MIME [2] (Multipurpose Internet Mail Extensions) has gained considerable acceptance and generated quite a bit of interest. We shall describe each of these formats in turn.

The MIME format is quite similar in many regards to the current Montage format and so we plan to convert the system over to MIME in the near future.

Current Format

To support the message format we wanted, we developed our own transport format based on SMTP. In this format, each individual message component was compressed, bundled, and converted to ASCII (since SMTP doesn't support binary message transmission). The conversion of the individual message components into a single transmittable block of data is called *packing*. This block of data then was transmitted as the body of a message, with the appropriate SMTP headers placed on the front of the message. Along with the message components themselves we transmit a *Table of Contents* (or TOC) which describes how to *unpack* the message body into its individual components, and the relationship of those components to one another.

In the current implementation of Montage, the TOC is a simple ASCII file which has a single line per component, and specifies the component name, whether it is a primary or attachment component, the relative position of the component in the mail message, the medium type, and compression type. The system packs and unpacks messages transparently to the user.

MIME

In many regards the MIME format is quite similar to the current Montage format and, since it will be supported on many more platforms than the current Montage format, we plan to convert the system to MIME in the near future.

MIME provides support for multipart messages in which each part contains the data for that part and specifies the type and encoding format for the data. The information in the MIME subparts is sufficient to allow for unencoding and decompressing Montage components, but there is no provision in MIME to specify any type of structural information about messages, such as component layout. Therefore we will transmit the Montage TOC as a separate MIME subpart so that Montage mailers can display the messages with all their structural connections, while other MIME-compliant mailers will display Montage messages in a linear layout.

IMPLEMENTATION NOTES AND STATUS

The current version of Montage is implemented using X11R4 and Motif 1.1. The code is approximately 40,000 lines of ANSI C.

This project was begun in late 1990 and resulted in a prototype implementation (based on the HP Widget Set) that demonstrated the basic concepts of the system (extensibility, external handlers, and so on) but was somewhat lacking in the features necessary to convince users to use the system on a day to day basis. In September of 1990 we began a reimplementation of the system based on Motif 1.1 and added a number of useful features.

Most of the features found in Montage are, of course, those found in any conventional non-multimedia mail system, such as folders, support for aliases, saving messages, replying to messages, and so on.

There are several other features specific to Montage have were incorporated into the Motif version though. Perhaps most important is the on-line configuration system. The prototype version of Montage used the X Resource Database to define media tags and map them to handlers. We wanted to separate the configuration of the system into appearance customizations (which would be handled by the X Resource Database mechanisms) and the more Montage-specific tag/handler mappings. One of the primary reasons for this was that we wanted users to be able to establish new mappings between tags and handlers without having to modify their X resource defaults.

Thus, we defined a format for Montage configuration files that contains information about tag-handler mappings. One side benefit of this choice is that we can provide a function in Montage to automatically rewrite the configuration file; this would have been awkward had we continued to use the X Resource Database because rewriting resource files would have lost any comments contained in the files (and thus customizing Montage would have erased comments on other applications' defaults).

Figure2 shows a Montage configuration panel for changing tag-handler mappings. Clicking the Accept button changed the preferences for the current session only. Clicking Save causes Montage to rewrite its own configuration file. These configuration files are, by the way, simple ASCII files and are very human-readable. The automatic configuration mechanisms simply provide an easy-to-use tool for customization by novice users.

In addition to the basic handlers mentioned earlier (editor, viewer, and renderer), Montage also keeps some other

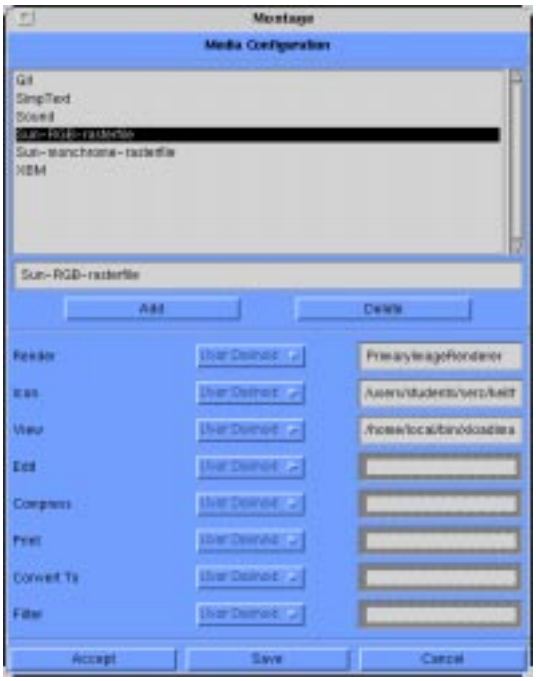

FIGURE 2. The Media Configuration Panel

information in its configuration database on a per-tag basis. These include:

- **Compressor** Denotes the type of compression used on the tag in question (for example, LZW for text components, JPEG for images).
- **Icon** The file to use for the icon image when a medium of the specified tag is used as an attachment.
- **Printer** A handler for printing the specified medium.
- **Filter** A handler for filtering the medium into text for display on a dumb terminal.
- **ConvertTo** The name of a tag to try to convert this medium into whenever it is encountered.
- **Converter** A handler which is used to perform the conversion to the new media type.

This information allows users to change the behavior of Montage in a number of ways. The method of compression can be changed to suit the particular medium being sent, icons can be chosen on a per-medium basis, handlers can be specified for printing and filtering, and so on.

A note on the converter mechanisms. Some media which are received may not be in a format which the reader can view. The converter mechanisms provides a means for Montage to automatically perform type conversions to a new medium which can be viewed (that is, for which a renderer or viewer is defined). Converters can be chained to any arbitrary level, and Montage can detect and break cycles in the converter graph.

CONCLUSIONS, CAVEATS, AND FUTURE DIRECTIONS

We feel that the current implementation of Montage serves to illustrate some useful concepts that are important for generalizable, flexible electronic mail systems. We view the current system as an “advanced prototype;” that is, the system implements a number of nice features but it is not commercial-quality software.

Currently, Georgia Tech is involved with licensing negotiations with several companies which wish to take Montage and turn it into a commercializable product. These companies are interested in adding features (such as security) which we are ill-equipped to do because of time and money constraints. Nevertheless, we would like to see Montage released into the community as freely distributable software even if the negotiations succeed.

In the area of future directions, we have several goals for Montage. The most important goal is support for the MIME standard for Internet multipart mail messages. We are also interested in exploring the domain of dynamic mail messages to a greater degree. In the current system, attachments are spatially collocated with the primary components they reference. We are interested in possibly exploring a time-based connection in which the main message body would be a dynamic component, such as video, and the attachments would be tied to a certain time point in the video and would scroll by at appropriate times.

While the concept of renderers provides a powerful extensibility mechanism, the current implementation leaves much to be desired, both in terms of flexibility and ease of implementation. We would like to experiment with more complex Montage-to-renderer protocols, perhaps using some sort of RPC-based mechanism. This would provide a greater degree of renderer control from within Montage (to support, for example, VCR-style controls on a video component).

We are also interested in the use of extension languages which could be bundled with Montage to give it even greater power. Such a system would allow high-level interpretted components to be sent as message components.

ACKNOWLEDGEMENTS

Thanks to our sponsors for this work, BellSouth and the Georgia Tech Advanced Technology Development Center. This work would not have been possible without their generous support.

I would also like to thank Tom Rodriguez and Jens Kilian for the significant amounts of time and energy they have devoted to the development of this system.

REFERENCES

[1] Nathaniel Borenstein. A Multimedia Message System for Andrew, in *Proceedings of USENIX Winter Conference*, February 1988.

[2] Nathaniel Borenstein, and Ned Freed. *MIME: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, Internet Draft.

[3] Barbara Chalfonte, Robert Fish, and Robert Kraut. Expressive Richness: A Comparison of Speech and Text as Media for Revision, in *Proceedings of ACM SIGCHI Conference*, 1991.

[4] David Crocker. *Standard for the Format of ARPA Internet Text Messages*, Internet Request for Comment (RFC) 822, August 13, 1982.

[5] Keith Edwards, *The Design and Implementation of the Montage Multimedia Mail System*, Technical Report GIT-SERC-90/04, April, 1990.

[6] Keith Edwards, The Design and Implementation of the Montage Multimedia Mail System, in *Proceedings of IEEE Conference on Communication Software* (TriComm), April 1991.

[7] S. Guastello, M. Traut, and G. Korienek. Verbal Versus Pictorial Representations of Objects in a Human-Computer Interface, in *International Journal of Man-Machine Studies*, July 1989.

[8] Brenda Laurel, Tim Oren, and Abbe Don. Issues in Multimedia Interface Design: Media Integration and Interface Agents, in *ACM SIGCHI Proceedings*, 1990.

[9] Jakob Nielsen. *Hypertext and Hypermedia*, Academic Press Inc., 1990

[10] G. Rohr. Using Visual Concepts, in *Visual Languages*, S. Chang, T. Ichikawa, and P. Ligomenides, eds., Plenum Press, 1986.

[11] W. Stallings. *Handbook of Computer Communications Standards, Volume 3: Department of Defense (DoD) Protocol Standards*. Macmillan, 1987.